Proceedings of the 41st IEEE
Conference on Decision and Control
Las Vegas, Nevada USA, December 2002

Jitterbug: A Tool for Analysis of Real-Time Control Performance

Bo Lincoln and Anton Cervin

Department of Automatic Control
Lund Institute of Technology
Box 118, SE 221 00 Lund, Sweden
{lincoln,anton}@control.lth.se

Abstract

The paper presents JITTERBUG, a MATLAB-based toolbox for real-time control performance analysis. The control system is described using a number of connected continuous-time and discrete-time linear systems driven by white noise. The control performance is measured by a continuous-time quadratic cost function. A stochastic timing model is used to describe when the different discrete-time systems are updated during the control period. Building different models, the tool makes it easy to investigate how the control performance is affected by e.g. delay, jitter, lost samples, aborted computations, and jitter compensation. Aperiodic and multi-rate controllers may also be studied. The tool is also capable of computing the spectral densities of the different signals in the system.

1. Introduction

Controllers are often designed with little regard for the real-time implementation. In the case of continuous-time design, it is typically assumed that the controller can be subsequently discretized and executed at a sufficiently high frequency. In the case of discrete-time design, it is commonly assumed that the computing platform can provide deterministic sampling and that the task execution will introduce negligible or at least constant computational delay.

In systems with limited computing resources (e.g. embedded control systems), however, a combination of slow sampling and other timing problems may lead to significant performance degradation. Often, the controller is implemented as a task in a (more or less real-time) operating system, and the task scheduling can introduce additional delays as well as sampling and actuation jitter (depending on how the I/O is implemented). In real-time operating systems which enforces hard deadlines, a control task may be aborted before it has finished its computations and produced a control signal. Networked control systems are another source of timing problems. The network can introduce delay and jitter, and messages (measurement or control signals) may be lost.

To achieve good performance in systems with limited computer resources, the constraints of the implementation must be taken into account at design time. Typically, trade-offs between different activities in the system must be made. For instance, boosting the priority of one task will improve its responsiveness but may introduce delay and jitter in other tasks. The periods of all tasks must be chosen such that the CPU is not overloaded, and so on. Having a quality-of-service measure which takes the timing effects into account can be a help when allocating system resources to the different tasks.

This paper presents a MATLAB-based toolbox called JITTERBUG which facilitates the computation of a quadratic performance criterion for a linear control system under various timing conditions. The tool helps to quickly assert how sensitive a control system is to delay, jitter, lost samples, etc, without resorting to simulation. The tool is quite general and can also be used to investigate for instance jitter-compensating controllers, aperiodic controllers, and multi-rate controllers. The toolbox is built upon well-known theory. Its main contribution is to make it easy to apply this type of stochastic analysis to a wide range of problems. The toolbox and a reference manual are available at `http://www.control.lth.se/~lincoln/jitterbug/`

The analysis in this paper builds on jump linear systems, which were first studied in [3]. Discrete-time jump linear systems are treated in e.g. [2]. Linear-quadratic analysis and control of systems with random network delays are studied in [4]. An alternative to analysis is simulation. The MATLAB/Simulink-based tool TRUETIME [1] can be used for detailed co-simulation of plant dynamics, control task execution, and real-time scheduling of CPU and network.

0-7803-7516-5/02/$17.00 ©2002 IEEE

2. System Description

In JITTERBUG, a control system is described by two parallel models: a signal model and a timing model. A simple model of a computer-controlled system is shown in Figure 1. The plant is described by the continuous-time system G, and the controller is described by two discrete-time systems, H_1 and H_2. The system H_1 could for instance represent a periodic sampler, while H_2 could represent the computation and actuation of the control signal. The associated timing model says that, at the beginning of each control period, H_1 should first be executed (updated). Then there is a (possibly random) delay τ_1 until H_2 is executed. This simple model could be used to investigate for instance the impact of delay and jitter on control performance. We will return to this example in Section 4.1.

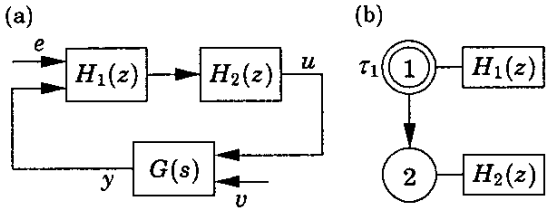

Figure 1: A simple JITTERBUG model of a computer-controlled system: (a) signal model, and (b) timing model.

2.1 Signal Model

The signal model consists of a number of interconnected continuous-time and discrete-time linear systems driven by white noise.

A *continuous-time system* is described by

$$\dot{x}_c(t) = Ax_c(t) + Bu(t) + v_c(t)$$
$$y(t) = Cx_c(t)$$

where A, B, and C are constant matrices, and v_c is a continuous-time white noise process with covariance[1] R_{1c}. The cost of the system is defined as

$$J_c = \lim_{T\to\infty} \frac{1}{T} \int_0^T x_c^T(t) Q_c x_c(t)\, dt$$

where Q_c is a positive semidefinite matrix.

A *discrete-time system* is described by

$$x_d(t_{k+1}) = \Phi x_d(t_k) + \Gamma u(t_k) + v_d(t_k)$$
$$y(t_k) = Cx_d(t_k) + Du(t_k) + e_d(t_k)$$

[1] By this we mean that v_c has the spectral density $\phi(\omega) = \frac{1}{2\pi} R_{1c}$.

where Φ, Γ, C, and D may be time-varying matrices (see below). The covariance of the discrete-time white noise processes v_d and e_d is given by

$$R_d = \mathbf{E} \begin{pmatrix} v_d(t_k) \\ e_d(t_k) \end{pmatrix} \begin{pmatrix} v_d(t_k) \\ e_d(t_k) \end{pmatrix}^T$$

The input signal u is sampled when the system is updated, and the output signal y is held between updates. The cost of the system is defined as

$$J_d = \lim_{T\to\infty} \frac{1}{T} \int_0^T x_d^T(t) Q_d x_d(t)\, dt$$

where Q_d is a positive semidefinite matrix. Note that the update instants t_k need not be equidistant in time, and that the cost is defined in continuous time.

The *total system* is formed by appropriately connecting the inputs and outputs of a number of continuous-time and discrete-time systems. Throughout, MIMO formulations are allowed, and a system may collect its inputs from a number of other systems. The total cost to be evaluated is summed over all continuous-time and discrete-time systems:

$$J = \sum J_c + \sum J_d \qquad (1)$$

2.2 Timing Model

The timing model consists of a number of timing nodes. Each node can be associated with zero or more discrete-time systems in the signal model which should be updated when the node becomes active. At time zero, the first node is activated. The first node can be declared to be *periodic* (indicated by an extra circle in the illustrations), which means that the execution will restart in this node every h seconds. This is useful to model periodic controllers and also simplifies the cost calculations a lot, see Section 3.3.

Each node is associated with a time delay τ which must elapse before the next node can become active. (If unspecified, the delay is assumed to be zero.) The delay can be used to model computational delay, transmission delay in a network, etc. A delay is described by a discrete-time probability density function

$$P_\tau = [\, P_\tau(0) \quad P_\tau(1) \quad P_\tau(2) \quad \ldots \,]$$

where $P_\tau(i)$ represents the probability of a delay of $i\delta$ seconds. The time grain δ is a constant which is specified for the whole model.

Node- and Time-Dependent Execution The same discrete-time system may be updated in several timing nodes. It is possible to specify different update equations (i.e. different Φ, Γ, C and D matrices) in

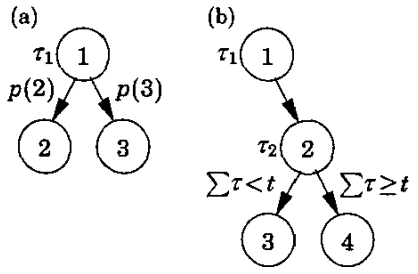

Figure 2: Alternative execution paths: (a) random choice of path, and (b) choice of path depending on the total delay from the first node.

the different cases. This can be used to model e.g. a filter where the update equations look different depending on whether a measurement value is available or not. An example of this type is given in Section 4.2.

It is also possible to make the update equations depend on the time since the first node became active. This can be used to model e.g. jitter-compensating controllers.

Alternative Execution Paths For some systems, it is desirable to specify alternative execution paths (and thereby multiple next nodes). In JITTERBUG, two such cases are possible to model, see Figure 2:

(a) A vector n of next nodes can be specified with a probability vector p. After the delay, node $n(i)$ will be activated with probability $p(i)$. This can be used to model e.g. a sample being lost with some probability.

(b) A vector n of next nodes can be specified with a time-vector t. If the total delay since the first node exceeds $t(i)$, node $n(i)$ will be activated next. This can be used to model e.g. time-outs and different compensation schemes.

Periodic vs Aperiodic Systems For periodic systems (i.e., for systems where the first timing node is periodic), the cost J can be calculated algebraically. The solver is fast and produces an exact solution. For aperiodic systems, the cost must be computed iteratively until it converges (if ever). From this point of view, periodic systems are clearly preferable.

In periodic systems, the execution is preempted if the total delay $\sum\tau$ in the system exceeds the period h. Any remaining timing nodes will be skipped. This models a real-time system where hard deadlines (equal to the period) are enforced and the control task is aborted at the deadline.

An aperiodic system can be used to model a real-time system where the task periods are allowed to drift if there are overruns. It could also be used to model e.g. a controller which samples "as fast as possible" instead of waiting for the next period.

3. Internal Workings

Inside JITTERBUG, the states and the cost are considered in continuous time. The inherently discrete-time states, e.g. in discrete-time controllers or filters, are treated as continuous-time states with zero dynamics. This means that the total system can be written as

$$\dot{x}(t) = Ax(t) + w(t) \tag{2}$$

where x collects all the states in the system, and w is continuous-time white noise process with covariance $\widetilde{R}$. To model the discrete-time changes of some states as a timing node n is activated, the state is instantaneously transformed by

$$x(t^+) = E_n x(t) + e_n(t)$$

where e_n is a discrete-time white noise process with covariance W_n.

The total cost (1) for the system can be written as

$$J = \lim_{T\to\infty} \frac{1}{T} \int_0^T x^T(t)\widetilde{Q}x(t)\,dt \tag{3}$$

where $\widetilde{Q}$ is a positive semidefinite matrix.

3.1 Sampling the System

JITTERBUG relies on discretized time to calculate the variance of the states and the cost. No approximations are involved, however. Sampling the system (2) with a period of δ (the time-grain in the delay distributions) gives

$$x(k\delta + \delta) = \Phi x(k\delta) + v(k\delta) \tag{4}$$

where the covariance of v is R, and the cost (3) becomes

$$J = \lim_{N\to\infty} \frac{1}{N\delta} \sum_{k=0}^{N-1} \Big(x^T(k\delta)Qx(k\delta) + q \Big)$$

The matrices Φ, R, Q, and q are calculated as

$$\Phi = e^{A\delta}$$

$$R = \int_0^\delta e^{A(\delta-\tau)} \widetilde{R} e^{A^T(\delta-\tau)}\,d\tau$$

$$Q = \int_0^\delta e^{A^T t} \widetilde{Q} e^{At}\,dt$$

$$q = \mathrm{tr}\Big(\widetilde{Q} \int_0^\delta \int_0^\delta e^{A(t-\tau)} \widetilde{R} e^{A^T(t-\tau)}\,d\tau\,dt \Big)$$

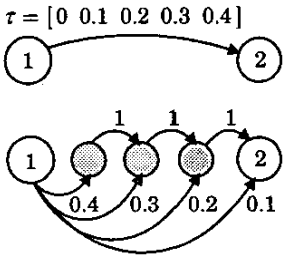

Figure 3: A random delay (above) modeled as a jump linear system (below), where the delay is represented by additional Markov nodes in between the timing nodes.

or, equivalently, from

$$\begin{pmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{pmatrix} = \exp\left(\begin{pmatrix} -A^T & \tilde{Q} \\ 0 & A \end{pmatrix} \delta \right)$$

and

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} = \exp\left(\begin{pmatrix} -A & I & 0 \\ 0 & -A & \tilde{R}^T \\ 0 & 0 & A^T \end{pmatrix} \delta \right)$$

so that

$$\begin{aligned} \Phi &= P_{22} \\ Q &= P_{22}^T P_{12} \\ R &= M_{33}^T M_{23} \\ q &= \operatorname{tr}\left(Q M_{33}^T M_{13}\right) \end{aligned}$$

3.2 Timing Representation

As time is discretized, we can transform the system description into a jump linear system, where the Markov state represents the current timing state of the system. Each timing node is represented by one Markov node. In between timing nodes additional Markov nodes representing the delay are inserted as illustrated in Figure 3.

Consider following one path in the Markov chain. For each node which is not a timing node, only the continuous states of the system change. In each time-step, they evolve as in (4), and thus the state covariance $P(k\delta) = \mathbf{E}\{x(k\delta)x^T(k\delta)\}$ evolves as

$$P(k\delta + \delta) = \Phi P(k\delta)\Phi^T + R$$

At each timing node n, the system is additionally transformed as in (3),

$$P(k\delta^+) = E_n P(k\delta) E_n^T + W_n$$

where W_n is the covariance of the discrete-time noise $e_n(k\delta)$ in node n. See Figure 4 for an illustration. Combining the above, we define Φ_n as

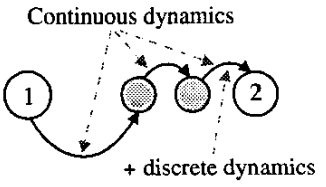

Figure 4: The continuous-time dynamics is active between all Markov nodes, whereas the discrete-time dynamics is activated only before a timing node.

$$\Phi_n = \begin{cases} \Phi & \text{if } n \text{ is not a timing node} \\ E_n \Phi & \text{if } n \text{ is a timing node} \end{cases}$$

and similarly R_n as

$$R_n = \begin{cases} R & \text{if } n \text{ is not a timing node} \\ E_n R E_n^T + W_n & \text{if } n \text{ is a timing node} \end{cases}$$

3.3 Calculating Variance and Cost

Now consider all possible Markov states simultaneously. Let $\pi_n(k\delta)$ be the probability of being in Markov state n at time $k\delta$, and let $P_n(k\delta)$ be the covariance of the state if the system is in Markov state n at time $k\delta$. Furthermore, let the transition matrix of the Markov chain be σ, such that

$$\pi(k\delta + \delta) = \sigma \pi(k\delta)$$

The state covariance then evolves as

$$P_n(k\delta + \delta) = \sum_i \sigma_{ni}\pi_i(k\delta)\left(\Phi_n P_i(k\delta)\Phi_n^T + R_n\right) \quad (5)$$

and the immediate cost at time $k\delta$ is calculated as

$$\frac{1}{\delta}\sum_n \pi_n(k\delta)\left(\operatorname{tr}\left(P_n(k\delta)Q\right) + q\right)$$

For systems without a periodic node, equation (5) must be iterated until the cost and variance converge. For periodic systems, the Markov state always returns to the periodic timing node every h/δ time steps. As equation (5) is affine in P, we can find the stationary covariance $P_1(\infty)$ in the periodic node by solving a linear system of equations. The total cost is then calculated over the timesteps in one period. The toolbox returns the cost $J = \infty$ if the system is not mean-square stable.

3.4 Calculating Spectral Densities

For periodic systems, the toolbox also computes the discrete-time spectral densities of all outputs as observed in the periodic timing node. The spectral density of an output y is defined as

$$\phi_y(\omega) = \frac{1}{2\pi}\sum_{k=-\infty}^{\infty} r_y(k)e^{-ik\omega}$$

The covariance function $r_y(k)$ is given by

$$r_y(k) = \mathbf{E}\left\{y(t)y^T(t+kh)\right\} = \mathbf{E}\left\{Cx(t)x^T(t+kh)C^T\right\}$$
$$= \mathbf{E}\left\{C\tilde{\Phi}^{|k|}x(t)x^T(t)C^T\right\} = C\tilde{\Phi}^{|k|}P_1(\infty)C^T$$

where $\tilde{\Phi}$ is the average transition matrix over a period, and $P_1(\infty)$ is the stationary covariance in the periodic node. The spectral density is returned as a linear system $F(z)$ such that $\phi_y(\omega) = F(e^{i\omega})$.

4. Examples

4.1 Delay and Jitter in a DVD Controller

Delays are a common problem in real-time control systems. In this example we examine the effect of randomly time-varying delays, or *jitter*. Generally, it is straight-forward to compensate for a constant delay, while jitter compensation is a harder issue.

In this example a DVD player focus control loop is considered. The JITTERBUG model of the system is shown in Figure 1. The plant $G(s)$ is given by a resonant sixth order continuous-time model of a DVD focus servo which has been obtained by system identification. The plant should be controlled by a discrete-time LQG controller with a sampling period of h seconds. The system $H_1(z) = 1$ represents a periodic sampler, while the system $H_2(z) = H(z)$ describes the control algorithm and the actuator. There is both process noise and measurement noise.

The sampler is executed at the beginning of each period. Then there is a random delay τ until the control signal is calculated and actuated. The LQG controller is designed to compensate for the mean delay. To study the combined effect of delay and jitter, the probability distribution for τ is given by a uniform distribution in the range $[m-j/2, m+j/2]$, where m is the mean delay and j is the jitter. An example script showing the MATLAB commands for a cost calculation is shown in Figure 5.

A plot showing the cost as a function of the mean delay and the jitter is given in in Figure 6. We can see that, in this case, the controller is quite sensitive to jitter. Naturally, the results are dependent on all model and design parameters. With JITTERBUG, it is easy to evaluate the effects of delays and jitter exactly for any parameters without resorting to simulations.

4.2 Lost Samples in Notch Filters

Cleaning signals from disturbances using e.g. notch filters is important in many applications. In some cases these filters are very sensitive to lost samples due to the very narrow-band characteristics, and in real-time systems lost samples are sometimes inevitable. In this example JITTERBUG is used to evaluate the effects of this problem on different filters.

```
Ptau = [0 1 1 1 0]/3;          Define delay distribution
load G;                  Load state-space model of the plant
H1 = tf(1,1,-1);                        Define the sampler
H2 = lqgdesign(G,Q,R1,R2,h,m);  Design the controller
N = initjitterbug(delta,h); Set time-grain and period
N = addtimingnode(N,1,Ptau,2);  Define timing node 1
N = addtimingnode(N,2);         Define timing node 2
N = addcontsys(N,1,G,3,Q,R1,R2);              Add plant
N = adddiscsys(N,2,H1,1,1);    Add sampler to node 1
N = adddiscsys(N,3,H2,2,2);  Add controller to node 2
N = calcdynamics(N);        Calculate internal dynamics
J = calccost(N);                      Calculate the cost J
```

Figure 5: An example MATLAB script showing the commands for a JITTERBUG cost calculation.

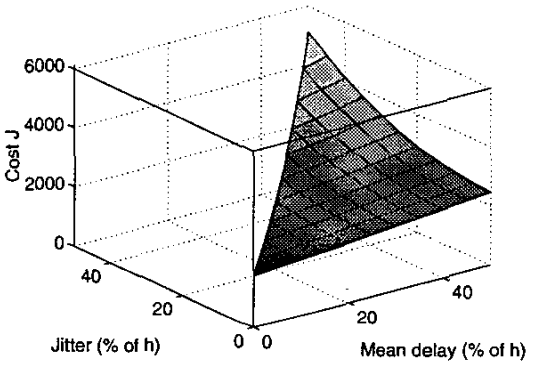

Figure 6: The cost of delay and jitter in the control loop. The controller is designed assuming a constant delay equal to the mean delay.

The setup is as follows. A good signal x (modeled as low-pass filtered noise) is to be cleaned from an additive disturbance e (modeled as band-pass filtered noise), see the signal spectra in Figure 7. An estimate $\hat{x}$ of the good signal should be found by applying a digital notch filter with the sampling interval $h = 0.1$ to the measured signal $x+e$. Unfortunately, a fraction p of the measurements are lost.

A JITTERBUG model of the system is shown in Figure 8. The signals x and e are generated by white noise being filtered through the continuous-time systems G_1 and G_2. The digital filter is represented as two discrete-time systems: *Samp* and *Filter*. The good signal is buffered in the system *Delay* and is compared to the filtered estimate in the system *Diff*. In the timing model, there is a probability p that the *Samp* system will not be updated. In that case, it is possible to execute an alternate version, *Filter*(2), of the filter dynamics.

Two different filters are compared. The first filter is

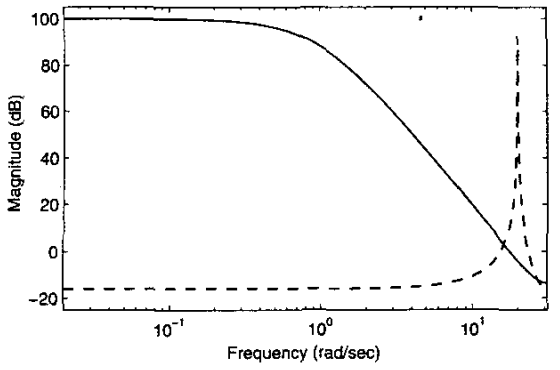

Figure 7: The spectral densities of the good signal x (full) and the disturbance e (dashed).

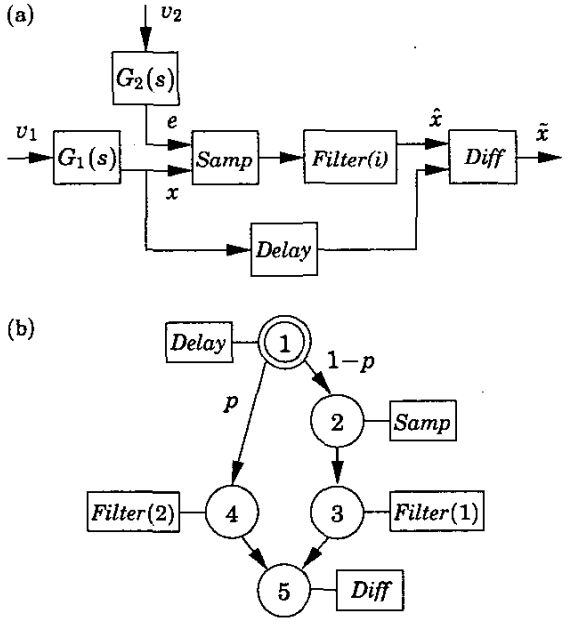

Figure 8: JITTERBUG model of the notch filter: (a) signal model, (b) timing model.

an ordinary second-order notch filter with two zeros on the unit circle. The same update equations are used regardless if a sample is available or not. The second filter is a second-order Kalman filter based on a simple model of the signal dynamics. In the case of lost samples, only prediction is performed in the filter.

The spectral density of the estimation error $\tilde{x} = x - \hat{x}$ in the two filter cases is shown in Figure 9. It has been assumed that $p = 10\%$ of the samples are lost. It is seen that the ordinary notch filter performs well around the disturbance frequency while the lost samples introduce a large error at lower frequencies. The time-varying Kalman filter is less sensitive towards

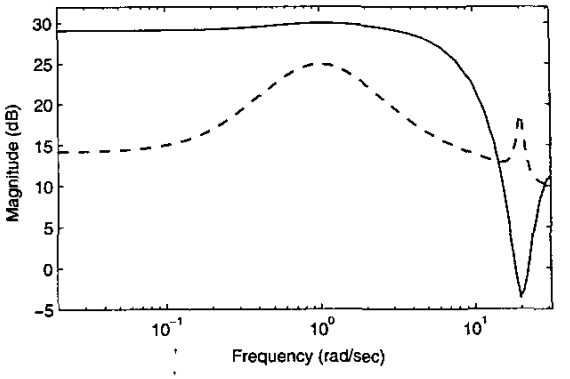

Figure 9: The spectral density of the error output $\tilde{x}$ when 10% of the samples are lost, using a notch filter (full) or a time-varying Kalman filter (dashed).

lost samples and has a more even error spectrum. Overall, the variance of the estimation error is about 40% lower in the Kalman filter case.

5. Conclusion

This paper has presented a MATLAB toolbox called JITTERBUG, which is used to compute a quadratic performance index for a real-time control system. The control (or signal processing) system is described using a number of continuous-time and discrete-time linear systems. A stochastic timing model with random delays is used to describe the execution of the system. Some of the things that can be investigated using the toolbox are delays, jitter, jitter compensation, lost samples, aborted computations, aperiodically sampled controllers, and multi-rate controllers.

References

[1] D. Henriksson, A. Cervin, and K.-E. Årzén. "Truetime: Simulation of control loops under shared computer resources." In *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

[2] Y. Ji, H. Chizeck, X. Feng, and K. Loparo. "Stability and control of discrete-time jump linear systems." *Control-Theory and Advanced Applications*, **7:2**, pp. 247–270, 1991.

[3] N. Krasovskii and E. Lidskii. "Analytic design of controllers in systems with random attributes, I, II, III." *Automation and Remote Control*, **22:9–11**, pp. 1021–1025, 1141–1146, 1289–1294, 1961.

[4] J. Nilsson. *Real-Time Control Systems with Delays*. PhD thesis ISRN LUTFD2/TFRT--1049--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, January 1998.